Exploring ChatGPT's capabilities for writing useful code

Vazquez Nunez Oscar, Matricule 000577005

14th May 2023

Contents

1 Introduction **2**
1.1 What are Large Language Models? 2
1.2 What is ChatGPT? 2

2 Methods **3**
2.1 Writing more complex programs 3
2.2 Writing simple programs 3
2.2.1 Recursive function for revealing squares 3
2.2.2 Printing the minesweeper board 3

3 Results **4**
3.1 Writing more complex programs 4
3.2 Writing simple programs 4
3.2.1 Recursive function for revealing squares 4
3.2.2 Printing minesweeper boards 4

4 Discussion **5**
4.1 Complex programs 5
4.2 Simple programs 5
4.3 The big picture 5
4.4 Methodological improvements and avenues for future research 6

5 Conclusion **6**

A Results for writing complex programs **7**
A.1 Dialogue 7
A.2 Screenshots of resulting outputs 10

B Results for writing complex programs **13**
B.1 Recursive function 13
B.1.1 Dialogue 13
B.1.2 Screenshots of relevant objects and outputs 15
B.2 Printing the minesweeper board 15
B.2.1 Dialogue 15
B.2.2 Screenshots 17

1 Introduction

In this paper, we explore and discuss the ability of chatbots powered by generative Large Language Models (LLMs), such as ChatGPT, to assist in programming tasks for the INFO-F106 course's first project: coding a game of minesweeper.

We look at ChatGPT's ability to assist with tasks such as writing simple functions and programs, and combining simple functions to write more complicated bodies of code, and we assess how practical its outputs are in producing a finished project. We find that while ChatGPT can produce very good output for many functions necessary for coding a game of minesweeper, even the more technical ones, it struggled to write code which adequately represents objects such as the game board to the player.

1.1 What are Large Language Models?

Generative LLMs are, in simple terms, input-output systems which are meant to handle an array of Natural Language Processing (NLP) tasks. The input is a string of text, whereas the output is a vector in $\mathbb{R}^v$ where v is the number of words, symbols, and portions of words that are in the LLM's 'vocabulary'. v_i represents the probability that the i-th element in the vocabulary is the 'right' next word[7]. Improved training algorithms, as well as scaling of the size of LLMs, have so improved them as to generate discussion of their potential to help and even replace humans in the production of articles, legal documents, and computer programs.

A paper[8] released in 2017 proposed transformers, a novel deep learning architecture based fully on attention. Attention algorithms allow for better tracking of long-term dependencies[5] in text, which is useful in language models as it allows them to relate words which are far from each other in a long text. For instance, an LLM using attention may more easily track which pronouns refer to which objects throughout a text. The paper showed that its architecture offered faster and more efficient training of models, and that those models outperformed previously existing models at certain translation tasks.

Later, OpenAI researchers used[5] the transformer architecture in generative pre-trained (GP) language models, which can train its model on unlabelled data[1]. That paper also explored fine-tuning GP models to perform specific tasks using manually labelled data, but it was the developement[6] of GPT-2[2] which indicated that GPT models could, when scaled up to have more parameters, perform downstream tasks[3] without any additional supervised training. GPT-3's release[2] showed that unsupervised LLMs could be further scaled up to improve at tasks such as performing arithmetic, translation and article-writing. Its release paper noted that GPT-3 could not only compete with state-of-the-art supervised models at various tasks, but could also generate samples from articles that were hard to distinguish from human works.

1.2 What is ChatGPT?

ChatGPT is a model similar to InstructGPT[4], in that the language model undergoes unsupervised training as all previous GPT models did, but is then fine-tuned with reinforcement learning from human feedback (RLHF), the expressed aim of which is to produce a model which can be benefit users by providing helpful and accurate responses. Since ChatGPT's release, perliminary research has been published which explores the model's ability to perform[3] and teach[1] tasks in computer science; this paper applies some suggestions and methodology from such existing research to the problem of coding a game of minesweeper.

---

[1] unlabelled data is data that hasn't been tagged with any descriptors. Labelled data is often useful in machine learning, as a model can compare its answers to the 'correct' label

[2] GPT stands for Generative Pre-trained Transformer

[3] downstream tasks can be thought of as tasks which the model was not explicitly trained for, but that we would like it to perform anyway

2 Methods

An ongoing study[3] published conversations between the authors and ChatGPT where the authors ask ChatGPT programming questions on six different topics, and assess the quality of its answers. We test it in two of those categories: **1. writing simple programs and functions** and **2. combining different functions to form more complex programs**[4]. For each of these categories, we choose prompts to query ChatGPT with which reflect and sometimes reproduce questions difficulties we faced when originally writing the project. We then ask it to explain or expand on its response to the original prompt, starting a dialogue. We use ChatGPT's programs and subsequent explanations and suggestions, as well as our own analysis and edits of its output code, to determine how practical or useful its responses would have been in completing the project.

2.1 Writing more complex programs

In this section, we ask ChatGPT to write out a whole program for a game of minesweeper using the prompt **'Could you write me a python code for running a game of minesweeper from the command line?'**
We then ask it follow-up questions about its program such as **'do you think this code has any bugs?'** and other queries more specific to its output. We then edit and debug its output code to determine how useful the code is to producing a finished minesweeper game.

2.2 Writing simple programs

In this section, we prompted ChatGPT to write different programs to complete specific tasks that were useful in the project. We specifically chose to query it on some of the tasks it performs less well on when attempting to write the entire game from scratch. Again, we follow up the original prompt with questions asking it to clarify or justify parts of its output, and test and modify its code to determine its viability as part of the project.

2.2.1 Recursive function for revealing squares

We gave ChatGPT the starting prompt **'Suppose you are writing a python program for a game of minesweeper. Write a recursive function that reveals the input square according to the rules of minesweeper.'**

We decided not to specify that the game must be played from the command window, as that is more or less irrelevant for writing this algorithm. The main interest in asking this question was to see whether and how ChatGPT would handle the most algorithmically complicated task involved in the project.

2.2.2 Printing the minesweeper board

We also prompt ChatGPT to print representations of the game board, both from a 'master' perspective which can see all the mines on the board, as well as from the perspective of the player. These questions are oriented towards finding out how well ChatGPT can turn objects in code into visual representations that humans can understand and interface with.

We ask ChatGPT: **'Suppose you are writing a program for a game of minesweeper that will be played from the command window, write a python program that takes in a matrix representing the game board, and prints a visual representation of the game board including all mines. Remember to represent mines with an 'X' in your code, and non-mines with a number which shows how many mines it has next to it.'**

---

[4] The programs that study asks ChatGPT to produce are somewhat easier than the corresponding 'simple' and 'complex' programs we ask it for

We then follow up with: **'Suppose you now need to print that same board to the player of the minesweeper game, which hides squares that haven't been revealed by the player, and hides mines. Write a python program for that.'**

3 Results

3.1 Writing more complex programs

In response to the starting prompt (A.1), ChatGPT produced a program which could print a starting square board for the game of minesweeper, and start a loop where players could input coordinates in the terminal to reveal squares. It could correctly (fig. 4) detect that the player had uncovered a mine and subsequently end a game with an appropriate message. From this we inferred that the board creation and mine placement code was probably working as intended. On the other hand, the program printed its own internal reference board out to the user, which displayed the location of all mines.

The program also suffered a 'recursion depth exceeded' (fig 5) error when we tried choosing a safe square. Fortunately we found the error and fixed it by simply adding a list of squares already covered by the recursive algorithm, which prevents it from infinitely revisiting squares. Unfortunately, the result was still not a playable game of minesweeper due to the confusing graphics. While the recursive function for revealing square was not resulting in a traceback anymore, the resulting boards (example in fig 6) didn't make sense given the rules of minesweeper.

Follow-up questions

ChatGPT originally produced a body of code which seemed incomplete with an obvious syntax error (fig. 1) at the end. When asked whether it thought its code had any bugs, it failed to detect any syntax errors and instead added some user input validation. Nevertheless, ChatGPT was able to complete its original code with correct syntax when asked directly about the error in the following dialogue. We also further prompted it to see if it would find the bug in its recursive algorithm; it reacted unexpectedly, offering an iterative algorithm instead of fixing its recursive one.

3.2 Writing simple programs

3.2.1 Recursive function for revealing squares

ChatGPT originally output a recursive function (B.1.1) with the right idea of revealing the chosen square an then revealing those around it if it if the revealed square has a chosen characteristic. However, it originally only represented whether the squares had been processed by its function, without providing information about what was around those squares (Fig 7). Additionally, the example provided by ChatGPT applied the algorithm a matrix representing a game board which doesn't make sense in the rules of minesweeper.

Follow-up questions

We then asked ChatGPT to rewrite its example, explicitly 'reminding' it to represent mines using X's and to represent safe squares by the number of mines around them. It reacted by rewriting its entire code, this time outputting an essentially correct function that has the same behaviour as the propagate_click function in our original code. Unfortunately, its example was not entirely correct due to its (fig 8) board not following minesweeper's rules. The function performed correctly on legal boards (fig 9 and fig 10).

3.2.2 Printing minesweeper boards

ChatGPT produced a program (B.2.1) that can adequately take an input board (fig 11) and print a minesweeper board where every square is revealed (fig 12). It also provided helpful explanations of its program. It did not format its board by, for example, numbering the rows and columns, but that was outside the scope of what it was prompted for.

ChatGPT performed less well when prompted to print a board for the player to see. Its code confuses squares that don't have mines on them with squares that the player has already revealed. It also didn't understand that revealed squares should contain numbers indicating how many mines are adjacent to them. The result (fig 13) is a board where mines are marked with asterisks, revealing their location to the player, while safe squares are completely unmarked. Whereas some of ChatGPT's other outputs in this paper could be easily modified to produce the intended result, the code for printing the player's board was extremely hard to get use out of.

4 Discussion

4.1 Complex programs

The first section in our methods describes us prompting ChatGPT to produce a full game of minsweeper. While ChatGPT wasn't able to produce a fully coherent program, some of the functions worked individually and could be used. Some others, like the recursive function for uncovering squares, could easily be debugged to serve their intended purpose. We suggest that the project could be completed more quickly with the aid of ChatGPT, for example by asking it for a full program, and then taking the functions which either work entirely or are easy to debug. It's less clear whether ChatGPT's input would have improved the quality of our project, or just decreased the time needed to complete it, though one qualitative benefit of asking it to write the full program was to be able to visualize what the the structure of the whole program might look like, even if some of the details aren't perfect.

While attempting to produce a completed game, ChatGPT struggled the most with the recursive function and with printing the game board our correctly. These results are what motivated us to prompt it to produce these functions individually, to see whether its output would improve in a simpler setting.

4.2 Simple programs

ChatGPT showed an ability to adapt to further prompts and instructions in the dialogue for the recursive function, and its second code output could likely have been substituted into our original program with only slight modifications. Its performance in producing and explaining the recursive function was much better when it wrote the function on its own, rather than as part of a larger body of code. This suggests that, while it maybe be interesting to see ChatGPT attempt to produce a full program, its usefulness may be improved by prompting it to produce subroutines one by one, as this leads to more accurate and less confusing code that is easier for the language model to explain and modify in accordance with further requests. Our assessment that ChatGPT could handle simpler programs well seems to support some of the perliminary results of the paper[3] which we modelled our question categrization after, as it also found that ChatGPT could usually produce and explain simpler programs which had few or no errors in them. It is however, worth cautioning[3] that ChatGPT's solutions and explanations can be wrong even if its 'tone' is confident.

Similarly, ChatGPT produced better code which it adapted more in line with our objectives when asked to only print the input board. However, it struggled just as much to print the board from the player's perspective as it did in the context of the larger program, even though it was given very explicit instructions on how to represent the board to the player.

4.3 The big picture

We observe a general trend in this paper's results. ChatGPT often produces correct or at least usable code for creating and handling backend objects. For example, it correctly wrote functions for randomly generating mines inside a board which it correctly represented as a matrix, and it wrote several recursive functions for revealing squares that were usable if not perfect.

However, it was much harder to get it to produce code that can represent these abstract objects in the necessary format for the user to actually play a game of minesweeper. We conjecture that ChatGPT's programs to create the 'backend' of a game of minesweeper would have been very helpful in completing those parts of the project more quickly, but that its code for representing those backend objects correctly to players was niether practically usable or well-explained by the chatbot.

4.4 Methodological improvements and avenues for future research

The methods of this paper could be improved by simply adding data to the very limited sample size. ChatGPT could be asked to solve a larger variety of different tasks to better understand its capabilities, but it may also be useful to give it the same prompt multiple times[5] to compare results and improve the reliability of what we think it answers to certain questions. Future research could possibly focus on exploring how ChatGPT can best be prompted to write programs that print out the data its manipulating to users correctly, for example in the context of printing a board for a game where certain items should be hidden from the user.

5 Conclusion

We examined ChatGPT's ability to produce and explain code with the aim of programming a game of minesweeper to be played through the command window. We conclude that ChatGPT's help could potentially have saved quite some time in thinking up and writing out certain functions and routines for the project. However, we also saw that some of its output needs to be edited heavily before working, and that moreover its results for printing out the minesweeper board to the player were almost totally impractical.

---

[5] It's worth noting that even though ChatGPT is an input-output system, it is *not* a function as it can produce different outputs to the same input

A Results for writing complex programs

A.1 Dialogue

`User: Could you write me a python code for running a game of minesweeper from the command line?`

`ChatGPT: Certainly! Here’s an example of a Python code for running a game of Minesweeper from the command line:`

```
import random

class MinesweeperGame:
    def __init__(self, size, num_mines):
        self.size = size
        self.num_mines = num_mines
        self.board = [[’-’ for _ in range(size)] for _ in range(size)]
        self.mines = self._place_mines()

    def _place_mines(self):
        mines = []
        positions = random.sample(range(self.size * self.size), self.num_mines)
        for pos in positions:
            row = pos // self.size
            col = pos % self.size
            mines.append((row, col))
            self.board[row][col] = ’X’
        return mines

    def _count_adjacent_mines(self, row, col):
        count = 0
        for i in range(-1, 2):
            for j in range(-1, 2):
                if i == 0 and j == 0:
                    continue
                new_row = row + i
                new_col = col + j
                if 0 <= new_row < self.size and 0 <= new_col < self.size and\
                 self.board[new_row][new_col] == ’X’:
                    count += 1
        return count

    def _reveal_cell(self, row, col):
        if self.board[row][col] != ’-’:
            return

        if (row, col) in self.mines:
            self.board[row][col] = ’X’
        else:
            count = self._count_adjacent_mines(row, col)
            self.board[row][col] = str(count) if count > 0 else ’-’
            if count == 0:
                for i in range(-1, 2):
                    for j in range(-1, 2):
                        new_row = row + i
                        new_col = col + j
                        if 0 <= new_row < self.size and 0 <= new_col < self.size:
                            self._reveal_cell(new_row, new_col)

    def _reveal_all_mines(self):
        for row, col in self.mines:
            self.board[row][col] = ’X’

    def play(self):
        while True:
            self._print_board()
            row = int(input(”Enter␣row␣(0␣to␣{}):␣”.format(self.size - 1)))
            col = int(input(”Enter␣column␣(0␣to␣{}):␣”.format(self.size - 1)))
            if (row, col) in self.mines:
                self._reveal_all_mines()
                self._print_board()
                print(”Game␣over!␣You␣stepped␣on␣a␣mine.”)
                break
            self._reveal_cell(row, col)
            if self._check_win():
                self._print_board()
                print(”Congratulations!␣You␣cleared␣all␣the␣mines.”)
                break

    def _check_win(self):
        for row in self.board:
            for cell in row:
                if cell == ’␣’:
                    return False
        return True

    def _print_board(self):
        print(”\nMinesweeper␣Board:\n”)
        print(”␣␣␣”, end=’’)
        for i in range(self.size):
            print(”{:2}”.format(i), end=’␣’)
        print(”\n”)
        for i, row in enumerate(self.board):
            print(”{:2}␣”.format(i), end=’’)
            print(’␣|␣’.join(row))
            if i < self.size - 1:
                print(”␣␣␣” + ”—+” * (self.size - 1) + ”—”)

if __name__ == ’__main__’:
    size = int(input(”Enter
```

User: do you think this code has any bugs?

ChatGPT:Upon reviewing the code, I’ve identified a potential bug. In the play() method, the user input for row and column is not validated to ensure it falls within the correct range. This can lead to a ValueError if the user enters an invalid input, such as a non-integer or a value outside the board’s range. To address this issue, you can modify the play() method to include input validation. Here’s an updated version:

```
def play(self):
    while True:
        self._print_board()
        try:
            row = int(input(”Enter␣row␣(0␣to␣{}):␣”.format(self.size - 1)))
            col = int(input(”Enter␣column␣(0␣to␣{}):␣”.format(self.size - 1)))
            if row < 0 or row >= self.size or col < 0 or col >= self.size:
                print(”Invalid␣input.␣Please␣enter␣a␣valid␣row␣and␣column.”)
                continue
        except ValueError:
            print(”Invalid␣input.␣Please␣enter␣a␣valid␣integer␣for␣row␣and␣column.”)
            continue

        if (row, col) in self.mines:
            self._reveal_all_mines()
            self._print_board()
            print(”Game␣over!␣You␣stepped␣on␣a␣mine.”)
            break
        self._reveal_cell(row, col)
        if self._check_win():
            self._print_board()
            print(”Congratulations!␣You␣cleared␣all␣the␣mines.”)
            break
```

With this modification, the program will handle invalid inputs gracefully and prompt the user to enter valid row and column values.

Please note that while this addresses the input validation issue, there may be other potential bugs or improvements that could be made in the code. It’s always a good idea to thoroughly test the code and handle edge cases to ensure its correctness and robustness.

User: what about the reveal_cell method? could you rewrite it so that recursion depth isn’t exceeded?

ChatGPT: Certainly! Here’s an updated version of the reveal_cell method that avoids exceeding the recursion depth by using an iterative approach instead of recursion:

```
def _reveal_cell(self, row, col):
    if self.board[row][col] != ’␣’:
        return

    stack = [(row, col)]
    while stack:
        curr_row, curr_col = stack.pop()
        if (curr_row, curr_col) in self.mines:
            self.board[curr_row][curr_col] = ’X’
        else:
            count = self._count_adjacent_mines(curr_row, curr_col)
            self.board[curr_row][curr_col] = str(count) if count > 0 else ’␣’
            if count == 0:
                for i in range(-1, 2):
                    for j in range(-1, 2):
                        new_row = curr_row + i
                        new_col = curr_col + j
                        if 0 <= new_row < self.size and 0 <= new_col < self.size and\
                   self.board[new_row][new_col] == '␣':
                            stack.append((new_row, new_col))
```

In this updated implementation, we use a stack to keep track of the cells that need to be revealed. We start with the initial cell, and as long as there are cells in the stack, we pop a cell from the sta check if it's a mine, update the board accordingly, and if it's not a mine and has no adjacent mines, we add its neighboring cells to the stack for further processing.

This iterative approach avoids exceeding the recursion depth, making the code more efficient and preventing potential stack overflow errors.

User: the end of your original code seems incomplete. How would you finish it

ChatGPT: Apologies for the incomplete code. Here's the complete version of the Minesweeper code, including the missing part:

```
if __name__ == '__main__':
    size = int(input("Enter␣the␣size␣of␣the␣board:␣"))
    num_mines = int(input("Enter␣the␣number␣of␣mines:␣"))

    game = MinesweeperGame(size, num_mines)
    game.play()
```

In the above code, we prompt the user to enter the size of the board and the number of mines they would like to have in the game. Then, we create an instance of the MinesweeperGame class with the specified size and number of mines. Finally, we call the play() method to start the game.

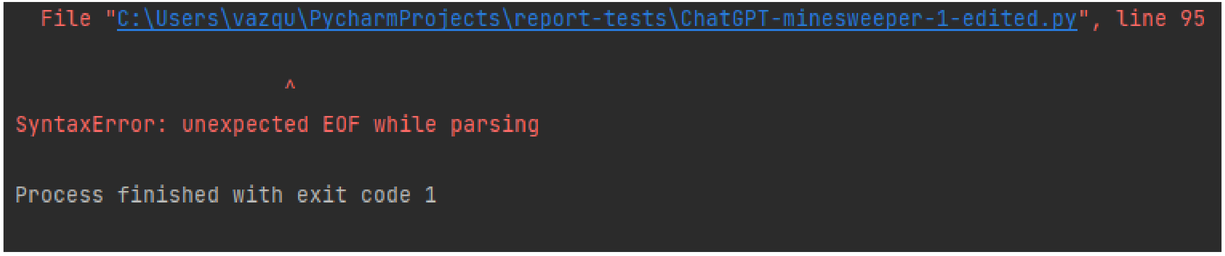
You can place the entire code, including the updated reveal_cell method and the missing part, in a Python file and run it from the command line to play the game of Minesweeper.

Make sure to save the file with a .py extension (e.g., minesweeper.py) and execute it by running python minesweeper.py in the command line after navigating to the directory where the file is located.

Feel free to adjust the code further if needed or add additional features to enhance the game.

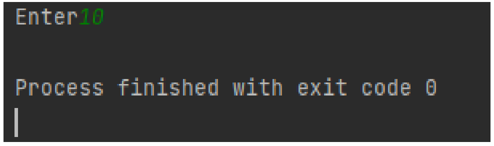
A.2 Screenshots of resulting outputs

Figure 1: Running ChapGPT's unedited code



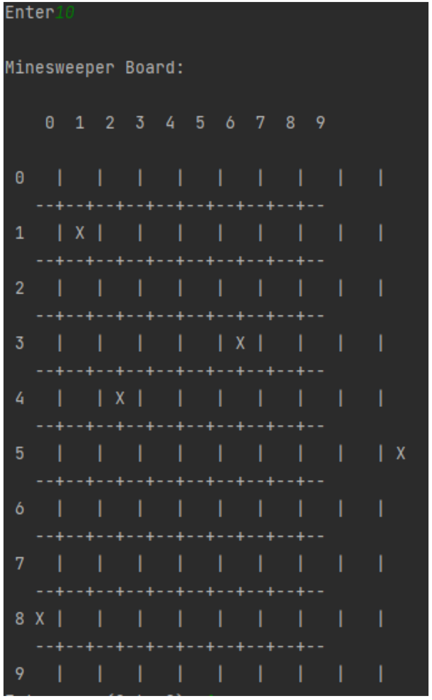
```
  File "C:\Users\vazqu\PycharmProjects\report-tests\ChatGPT-minesweeper-1-edited.py", line 95

                     ^
SyntaxError: unexpected EOF while parsing


Process finished with exit code 1
```

Figure 2: Results after fixed of syntax errors



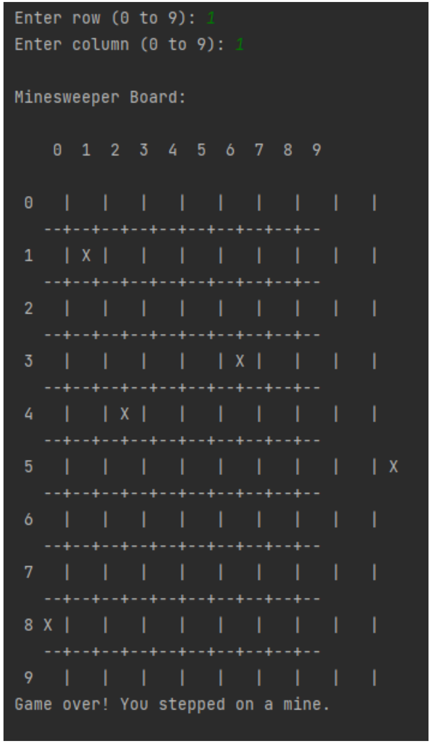
```
Enter10

Process finished with exit code 0
```

Figure 3: result after ChatGPT completed its code



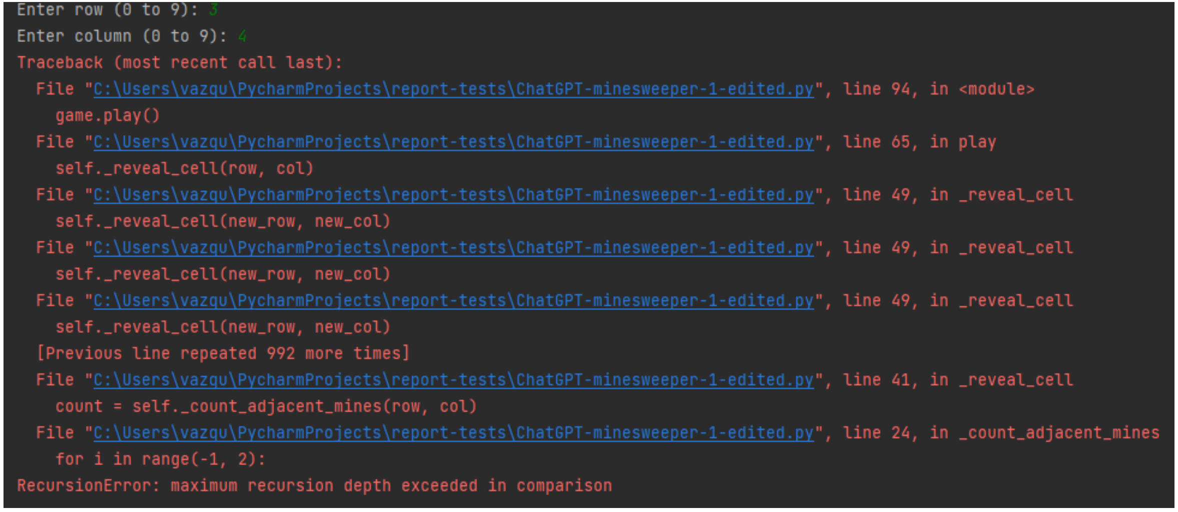
```
Enter10

Minesweeper Board:

   0  1  2  3  4  5  6  7  8  9

 0   |   |   |   |   |   |   |   |   |
   --+--+--+--+--+--+--+--+--+--
 1   | X |   |   |   |   |   |   |   |
   --+--+--+--+--+--+--+--+--+--
 2   |   |   |   |   |   |   |   |   |
   --+--+--+--+--+--+--+--+--+--
 3   |   |   |   |   | X |   |   |   |
   --+--+--+--+--+--+--+--+--+--
 4   |   | X |   |   |   |   |   |   |
   --+--+--+--+--+--+--+--+--+--
 5   |   |   |   |   |   |   |   |   | X
   --+--+--+--+--+--+--+--+--+--
 6   |   |   |   |   |   |   |   |   |
   --+--+--+--+--+--+--+--+--+--
 7   |   |   |   |   |   |   |   |   |
   --+--+--+--+--+--+--+--+--+--
 8 X |   |   |   |   |   |   |   |   |
   --+--+--+--+--+--+--+--+--+--
 9   |   |   |   |   |   |   |   |   |
```

Figure 4: result after choosing a mine



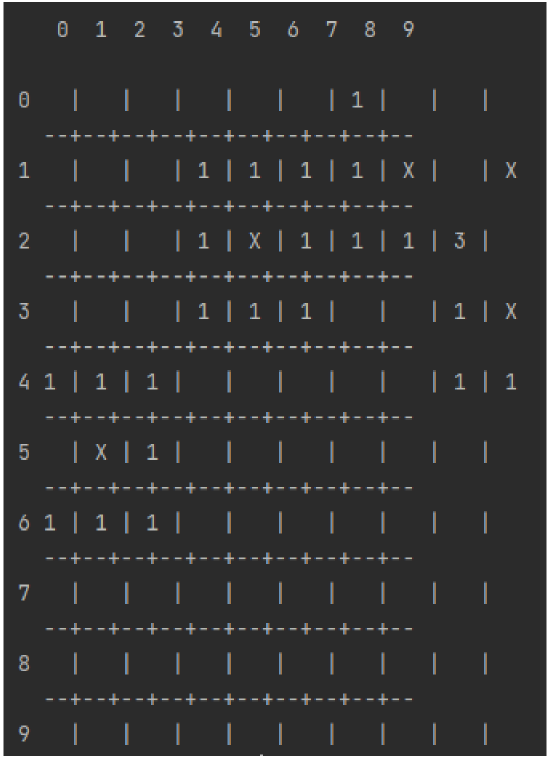
```
Enter row (0 to 9): 1
Enter column (0 to 9): 1

Minesweeper Board:

   0  1  2  3  4  5  6  7  8  9

 0   |   |   |   |   |   |   |   |   |
   --+--+--+--+--+--+--+--+--+--
 1   | X |   |   |   |   |   |   |   |
   --+--+--+--+--+--+--+--+--+--
 2   |   |   |   |   |   |   |   |   |
   --+--+--+--+--+--+--+--+--+--
 3   |   |   |   |   | X |   |   |   |
   --+--+--+--+--+--+--+--+--+--
 4   |   | X |   |   |   |   |   |   |
   --+--+--+--+--+--+--+--+--+--
 5   |   |   |   |   |   |   |   |   | X
   --+--+--+--+--+--+--+--+--+--
 6   |   |   |   |   |   |   |   |   |
   --+--+--+--+--+--+--+--+--+--
 7   |   |   |   |   |   |   |   |   |
   --+--+--+--+--+--+--+--+--+--
 8 X |   |   |   |   |   |   |   |   |
   --+--+--+--+--+--+--+--+--+--
 9   |   |   |   |   |   |   |   |   |
Game over! You stepped on a mine.
```

Figure 5: recursion error after choosing an empty square



```
Enter row (0 to 9): 3
Enter column (0 to 9): 4
Traceback (most recent call last):
  File "C:\Users\vazqu\PycharmProjects\report-tests\ChatGPT-minesweeper-1-edited.py", line 94, in <module>
    game.play()
  File "C:\Users\vazqu\PycharmProjects\report-tests\ChatGPT-minesweeper-1-edited.py", line 65, in play
    self._reveal_cell(row, col)
  File "C:\Users\vazqu\PycharmProjects\report-tests\ChatGPT-minesweeper-1-edited.py", line 49, in _reveal_cell
    self._reveal_cell(new_row, new_col)
  File "C:\Users\vazqu\PycharmProjects\report-tests\ChatGPT-minesweeper-1-edited.py", line 49, in _reveal_cell
    self._reveal_cell(new_row, new_col)
  File "C:\Users\vazqu\PycharmProjects\report-tests\ChatGPT-minesweeper-1-edited.py", line 49, in _reveal_cell
    self._reveal_cell(new_row, new_col)
  [Previous line repeated 992 more times]
  File "C:\Users\vazqu\PycharmProjects\report-tests\ChatGPT-minesweeper-1-edited.py", line 41, in _reveal_cell
    count = self._count_adjacent_mines(row, col)
  File "C:\Users\vazqu\PycharmProjects\report-tests\ChatGPT-minesweeper-1-edited.py", line 24, in _count_adjacent_mines
    for i in range(-1, 2):
RecursionError: maximum recursion depth exceeded in comparison
```

Figure 6: result after one 'sucessful' recursion



```
   0  1  2  3  4  5  6  7  8  9


0   |   |   |   |   |   | 1 |   |   |
  --+--+--+--+--+--+--+--+--+--
1   |   |   | 1 | 1 | 1 | 1 | X |   | X
  --+--+--+--+--+--+--+--+--+--
2   |   |   | 1 | X | 1 | 1 | 1 | 3 |
  --+--+--+--+--+--+--+--+--+--
3   |   |   | 1 | 1 | 1 |   |   | 1 | X
  --+--+--+--+--+--+--+--+--+--
4 1 | 1 | 1 |   |   |   |   |   | 1 | 1
  --+--+--+--+--+--+--+--+--+--
5   | X | 1 |   |   |   |   |   |   |
  --+--+--+--+--+--+--+--+--+--
6 1 | 1 | 1 |   |   |   |   |   |   |
  --+--+--+--+--+--+--+--+--+--
7   |   |   |   |   |   |   |   |   |
  --+--+--+--+--+--+--+--+--+--
8   |   |   |   |   |   |   |   |   |
  --+--+--+--+--+--+--+--+--+--
9   |   |   |   |   |   |   |   |   |
```

B Results for writing complex programs

B.1 Recursive function

B.1.1 Dialogue

User: Suppose you are writing a python program for a game of minesweeper. Write a recursive function that reveals the input square according to the rules of minesweeper.

ChatGPT: Certainly! Here's an example of a recursive function in Python that reveals a square in a game of Minesweeper:

```
def reveal_square(board, revealed, row, col):
    if revealed[row][col]:
        return

    revealed[row][col] = True
    if board[row][col] == 0:
        # Recursively reveal adjacent squares
        for i in range(-1, 2):
            for j in range(-1, 2):
                new_row = row + i
                new_col = col + j
                if 0 <= new_row < len(board) and 0 <= new_col < len(board[0]):
                    reveal_square(board, revealed, new_row, new_col)
```

This function takes three parameters: board, revealed, row, and col. The board represents the Minesweeper game board, where each element contains the value of the square (0 for empty, 1-8 for the number of adjacent mines, -1 for a mine). The revealed list keeps track of which squares have already been revealed.

The function checks if the square at the given row and column has already been revealed. If it has, the function returns. Otherwise, it marks the square as revealed and checks its value. If the value is 0, it recursively reveals all adjacent squares by calling itself on each adjacent square.

To use this function, you can create the initial game board and revealed list, and then call reveal_square to reveal a square:

```
# Example usage
board = [
    [0, 0, -1, 1],
    [0, 0, 1, 1],
    [0, 0, 1, -1],
    [1, 1, 1, 1]
]

revealed = [
    [False, False, False, False],
    [False, False, False, False],
    [False, False, False, False],
    [False, False, False, False]
]

reveal_square(board, revealed, 2, 1)
```

In this example, calling reveal_square(board, revealed, 2, 1) will reveal the square at row 2, column 1, as well as any adjacent squares with a value of 0.

User: Can you rewrite your example, using 'X' to represent mines on your board, and using numbers on squares that don't have mines. Remember that the numbers represent how many mines are adjacent to the square.

ChatGPT: Certainly! Here's an updated version of the code that uses 'X' to represent mines and numbers for squares without mines:

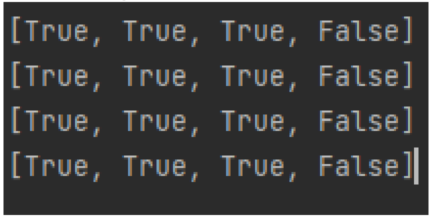
```python
def reveal_square(board, revealed, row, col):
    if revealed[row][col]:
        return

    revealed[row][col] = True
    if board[row][col] == 0:
        # Recursively reveal adjacent squares
        for i in range(-1, 2):
            for j in range(-1, 2):
                new_row = row + i
                new_col = col + j
                if 0 <= new_row < len(board) and 0 <= new_col < len(board[0]):
                    reveal_square(board, revealed, new_row, new_col)

# Example usage
board = [
    ['X', 'X', 0, 1],
    [2, 3, 1, 1],
    [0, 1, 'X', 'X'],
    [1, 'X', 2, 1]
]

revealed = [
    [False, False, False, False],
    [False, False, False, False],
    [False, False, False, False],
    [False, False, False, False]
]

reveal_square(board, revealed, 2, 1)

# Print the revealed board
for i in range(len(board)):
    for j in range(len(board[0])):
        if revealed[i][j]:
            print(board[i][j], end=' ')
        else:
            print('-', end=' ')
    print()
```

In this updated version, 'X' is used to represent mines on the board, and numbers (instead of 0) are used to indicate the number of adjacent mines for each square without a mine. The reveal_square function remains the same.

After calling reveal_square(board, revealed, 2, 1), the function will reveal the square at row 2, column 1, as well as any adjacent squares with a value of 0. Finally, the revealed board is printed to show the revealed squares with their respective values or ’X’ for mines.

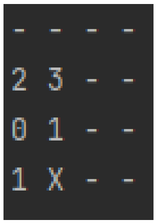
B.1.2 Screenshots of relevant objects and outputs

Figure 7: revealed list



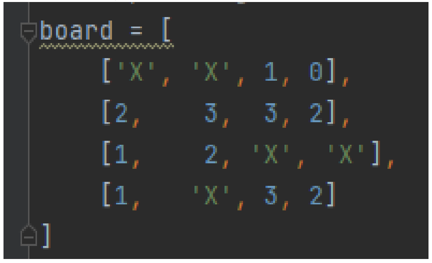
```
[True, True, True, False]
[True, True, True, False]
[True, True, True, False]
[True, True, True, False]
```

Figure 8: inaccurate board



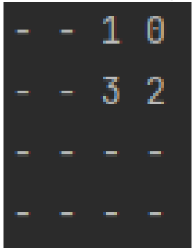
```
- - - -
2 3 - -
0 1 - -
1 X - -
```

Figure 9: correct board



```
board = [
    ['X', 'X', 1, 0],
    [2,    3,  3, 2],
    [1,    2, 'X', 'X'],
    [1,   'X', 3, 2]
]
```

Figure 10: resulting reveal



```
- - 1 0
- - 3 2
- - - -
- - - -
```

B.2 Printing the minesweeper board

B.2.1 Dialogue

User: Suppose you are writing a program for a game of minesweeper that will be played from the command window, write a python program that takes in a matrix representing the game board, and prints a visual representation of the game board including all mines. Remember to represent mines with an ’X’ in your code, and non-mines with a number which shows how many mines it has next to it.

ChatGPT: Certainly! Here’s an example of a Python program that takes a matrix representing the game board and prints a visual representation of the board, including mines represented by ’X’ and non-mines represented by the number of adjacent mines:

```
def count_adjacent_mines(board, row, col):
    count = 0
    for i in range(max(0, row - 1), min(row + 2, len(board))):
        for j in range(max(0, col - 1), min(col + 2, len(board[0]))):
            if board[i][j] == 'X':
                count += 1
    return count

def print_board(board):
    for row in board:
        print('␣'.join(row))

def reveal_board(board):
    revealed_board = [['␣' for _ in range(len(board[0]))] for _ in range(len(board))]

    for i in range(len(board)):
        for j in range(len(board[0])):
            if board[i][j] == 'X':
                revealed_board[i][j] = 'X'
            else:
                count = count_adjacent_mines(board, i, j)
                revealed_board[i][j] = str(count) if count > 0 else '␣'

    print_board(revealed_board)

# Example game board
game_board = [['␣', 'X', '␣', '␣'],
              ['X', '␣', '␣', 'X'],
              ['␣', '␣', 'X', '␣'],
              ['␣', 'X', '␣', '␣']]

reveal_board(game_board)
```

This program defines two functions: count_adjacent_mines and reveal_board. The count_adjacent_mines function takes the game board, row, and column as input and counts the number of adjacent mines. The reveal_board function creates a new board that reveals the contents of each cell, either an 'X' for a mine or the count of adjacent mines.

In the example, the game_board variable represents the initial state of the game board. When reveal_board(game_board) is called, it will print the visual representation of the game board with mines represented by 'X' and non-mines represented by the count of adjacent mines.

You can modify the game_board variable to represent different game scenarios or create your own board and pass it to the reveal_board function.

User: Suppose you now need to print that same board to the player of the minesweeper game, which hides squares that haven't been revealed by the player, and hides mines. Write a python program for that.

ChatGPT: Certainly! Here's an updated version of the Python program that prints the game board, hiding squares that haven't been revealed by the player and hiding mines:

```
def print_board(board, revealed):
    for i in range(len(board)):
        for j in range(len(board[0])):
            if revealed[i][j]:
                print(board[i][j], end='␣')
            else:
                print('*', end='␣')
        print()
```

```python
def reveal_board(board):
    rows = len(board)
    cols = len(board[0])
    revealed = [[False for _ in range(cols)] for _ in range(rows)]

    for i in range(rows):
        for j in range(cols):
            if board[i][j] == 'X':
                revealed[i][j] = False
            else:
                count = count_adjacent_mines(board, i, j)
                revealed[i][j] = True

    print_board(board, revealed)

# Example game board
game_board = [['-', 'X', '-', '-'],
              ['X', '-', '-', 'X'],
              ['-', '-', 'X', '-'],
              ['-', 'X', '-', '-']]

reveal_board(game_board)
```

`In this updated program, the print_board function takes an additional parameter called revealed, which is a 2D boolean array representing whether each cell has been revealed by the player. It prints the game board, hiding squares that haven't been revealed by the player and replacing them with an asterisk *. The reveal_board function initializes the revealed array based on the content of the board array, where mines are set to False and non-mines are set to True.`

`The example game_board remains the same, and when reveal_board(game_board) is called, it will print the game board, hiding squares that haven't been revealed and replacing mines with asterisks *.`

`You can modify the game_board variable to represent different game scenarios or create your own board and pass it to the reveal_board function.`

B.2.2 Screenshots

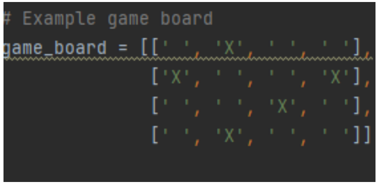


Figure 11: example board

```python
# Example game board
game_board = [[' ', 'X', ' ', ' '],
              ['X', ' ', ' ', 'X'],
              [' ', ' ', 'X', ' '],
              [' ', 'X', ' ', ' ']]
```

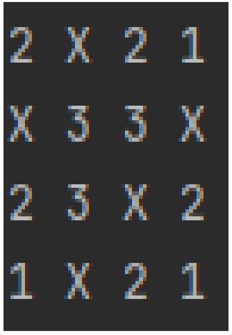


Figure 12: revealed board

```
2 X 2 1
X 3 3 X
2 3 X 2
1 X 2 1
```

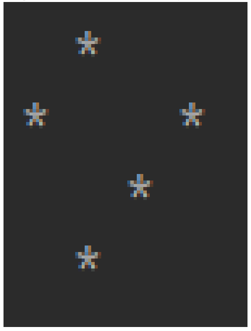


Figure 13: faulty reveal

```
  *
*     *
    *
  *
```

References

[1] Paramarshi Banerjee, Anurag Srivastava, Donald Adjeroh, Y. Ramana Reddy, and Nima Karimian. Understanding ChatGPT: Impact Analysis and Path Forward for Teaching Computer Science and Engineering. 4 2023.

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[3] James C. Davis, Yung-Hsiang Lu, and George K. Thiruvathukal. Conversations with ChatGPT about C Programming: An Ongoing Study. 3 2023.

[4] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.

[5] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training, 2018.

[6] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[7] Murray Shanahan. Talking about large language models, 2023.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.